ADG Best Practices – Projects

Source Control

All new ADG projects should utilize a git source control repository. The ADG uses Azure DevOps, formerly Visual Studio Team Services, for hosting the remote repositories of all ADG projects. These can be found at https://dev.azure.com/eur-io/. Each project has a corresponding Azure DevOps project, which can then contain multiple git repositories if needed.

Git Repository Structure

All projects will begin with a master branch, which should mirror the code that's in production. Other environments should also have their own branch to allow for continuous deployment, as well as, an easy way to update the code and release to that particular environment.

At a minimum, the first branch off of master should be develop and should mirror a staging environment. As features are developed, developers can then create branches off of develop for their assigned features, which can then be merged back into develop for testing in the staging environment.

Visual Studio Solutions

Using a standard approach to all Visual Studio Solutions allows ADG developers to have a seamless transition onto new projects. Each solution should target a similar set of projects with common folder structures. The standard naming convention for each project is the product's name followed by the project's purpose, like the following: *ADG.Domain*, *ADG.Service*, and *ADG.Web*. Small projects may opt to go for a single layer approach with just the Web project. Projects should utilize existing ADG Components to minimize duplication of feature development. The ADG Components are available in the ADG Template files for new projects, and the components are also available through the ADG Nuget Server hosted on Azure for existing projects.

Domain

The Domain project contains the database access layer of the application. The preferred database is an Azure SQL Database that is accessed via Entity Framework using the Code-First approach applying Code-First Migrations to build the database. If an existing database is being supported by the new application the Database-First approach may be necessary and is acceptable.

References

References in the Domain project should include, but are not limited to, EntityFramework, Newtonsoft.Json, System.ComponentModel.DataAnnotations, and their corresponding dependencies.

Structure

The Domain project should contain a Context folder that includes the class for the inherited DbContext used in Entity Framework, as well as, any initializers used for seeding the initial database. An interface for the DbContext is also necessary for injecting it into services.

If Entity Framework Code-First is being used the Domain project should contain a Migrations folder, and it should contain all of the Code-First Migrations.

The Domain project should contain a Models folder for all of the Entity Framework model classes, and each model class should be in its own file.

Service

The Service project contains the business logic. There will be a collection of Services that can inherit from a base EntityService and will be the only classes interacting with the Domain project's DbContext for accessing data. By implementing as much business logic as possible in the Service project, it allows it to be reused with any application on top of it in the future.

References

References in the Service project should include, but are not limited to, the Domain project, EntityFramework, Newtonsoft.Json, System.ComponentModel.DataAnnotations, AutoMapper, and their corresponding dependencies.

Structure

The Service project should contain a Framework folder that contains an abstract ServiceBase and/or EntityService to be inherited by each service in the project. It should also contain any other necessary abstract class or interface, other than those interfaces for each individual service, that is used as a framework for the rest of the project.

The Service project should contain an Interfaces folder to include an interface for each service created, so that it can be implemented by Dependency Injection.

The Service project should contain a Services folder that contains each service that inherits from the abstract base and has a corresponding interface. These services should all take an interfaced DbContext in the constructor that will be provided by Dependency Injection.

Web

The Web project should be a .NET Web Application. All new projects should use a minimum of .NET 4.5 with MVC 5, with an eventual adoption of .NET Core MVC.

References

References in the Web project should include, but are not limited to, the Service project, Newtonsoft.Json, System.ComponentModel.DataAnnotations, AutoMapper, SimpleInjector, and their corresponding dependencies

Structure

The Web project should contain the prototypical folder and file structure for a .NET Web Application. Areas can be utilized for splitting large areas of an application from one another. Controllers should all inherit an abstract base controller for any universal functionality, and an additional abstract base controller can be added to each area as well. Controllers should receive their necessary service through Dependency Injection (DI) using Simple Injector, the standard DI tool the ADG has adopted.

The AutoMapper library should be used for mapping from Service model objects to Web view model objects. The configuration for the mappings can be found in the AutoMapperConfig.cs file in the App_Start folder.

To help structure partial views, it's suggested to break the standard Shared folder into multiple sub folders, including DisplayTemplates, EditorTemplates, Layouts, and Partials. Remember to register these routes in the Razor View Engine.

Another important step is to ensure that sensitive keys/connection strings aren't stored in the web.config file. When using Azure, each Web App has an Application Settings panel that can store this data, so it never has to be in your application. Any local development settings that are not sensitive can remain in the web.config. If development spans across multiple developers, it may be desirable to put connection strings in a separate file that is ignored by git and then referenced in the web.config file instead of constantly overwriting each developer's connection strings.